An Efficient Scheme for Preserving Confidentiality in Content-Based Publish-Subscribe Systems

Jun Li Chenghuai Lu Weidong Shi
College of Computing, Georgia Institute of Technology

January 20, 2004

Abstract

Content-based publish-subscribe is an efficient communication paradigm that supports dynamic, many-to-many data dissemination in a distributed environment. A publish-subscribe system deployed over a wide-area network must handle information dissemination across distinct authoritative domains and heterogeneous platforms. Such an environment raises serious security concerns. This paper describes a practical scheme that preserves confidentiality against eavesdroppers for private content-based publish-subscribe systems over public networks. In this scheme, publications and subscriptions are encrypted, while the publish-subscribe infrastructure is able to make correct routing decisions based on encrypted publications and subscriptions. Plaintexts are not revealed in the infrastructure for the purpose of security and efficiency. This scheme efficiently supports interval-matching as a predicate function for subscriptions. The security of this scheme is analyzed, and further improved by several techniques.

Keywords: Content-based publish-subscribe system, confidentiality, interval-matching, prefix-matching, prefix-preserving encryption/decryption.

1 Introduction

This paper describes an efficient scheme that preserves confidentiality against eavesdroppers for private content-based publish-subscribe systems over public networks. The content-based publish-subscribe (*pub-sub*) model [23] is an efficient communication paradigm that supports dynamic, many-to-many data dissemination in a distributed environment. One pioneering work on content-based pub-sub is the IBM Gryphon project [3, 4, 21, 22], which has been deployed over the Internet for real-time sports score distribution at the US Tennis Open, Ryder Cup, and Australian Open, and for monitoring and statistics reporting at the Sydney Olympics [1].

In general, a pub-sub system is a data dissemination system that consists of *information providers*, who publish *events* (the unit of information in this paper) to the system, *information consumers*, who subscribe to particular categories of events, and a network of *brokers*, who are responsible for routing events from publishers to interested subscribers. In this paper the broker network is often referred to as the *infrastructure* of a pub-sub system. In a content-based pub-sub

system, an *event schema* specifies the types of information carried in an event and a language is defined for subscribers to describe events that are of interest to them. Such description is often defined as a *predicate function* over the event content. With the events from publishers and the predicate functions from subscribers, the broker network is responsible for dispatching the events from publishers to interested subscribers.

One example of content-based pub-sub system is the stock quote pub-sub system described in [3, 4, 21, 22]. A stock event schema may include the stock symbol, price, and volume, of string, dollar, and integer types respectively. A subscription is defined as a predicate over those three *attributes*, such as (symbol = "ATT") and ($20 $\leq$ price $\leq$ $80) and (2000 $\leq$ volume $\leq$ 4000 or 20000 $\leq$ volume $\leq$ 40000). Upon receipt of a stock event, the broker network performs an exact-matching on the stock symbol and interval-matchings on the price and volume according to the subscription and decides whether the event should be routed to the subscriber. For example, an event in which the symbol is "ATT", the price is $50 and the volume is 3000 will be routed to the subscriber through the broker network.

The security concerns of content-based pub-sub systems have been pointed out in [26], but most of the solutions remain open problems. One major security issue is confidentiality.

- When information being published contains sensitive content, publishers and subscribers may wish to keep the event content secret. For example, an application distributing premium stock reports may want to make sure that only interested subscribers can access the data. This is referred to as *information confidentiality* [26].
- User subscriptions can reveal sensitive information about the user, in which case the subscriber may wish to keep the subscriptions private. For example, in the stock quote pub-sub system, subscribers may not want others to know which stocks they are interested in. This is referred to as *subscription confidentiality* [26].

Corporations with many sites often build private networks over public networks. Therefore the broker network may not be well protected. The broker network may operate on top of a heterogeneous network infrastructure that is poorly protected at some places. For example, the broker network may have links over a wireless network that often has weak protection. Therefore, the plaintexts of publications/subscriptions need to be protected from the infrastructure. Meanwhile the content-based pub-sub system should still preserve its functionality, and ideally work as efficiently as before.

One simple way to preserve the confidentiality is the point-to-point encryption/decryption approach. The publishers encrypt every published event using an established encryption scheme. Then event brokers will decrypt the event, examine the content, and route the event to the interested subscribers after re-encrypting it. There are several drawbacks with this approach. First, the event is decrypted and encrypted at each broker on the path from the publisher to the interested subscriber. This is very inefficient. Second, brokers are not necessarily well protected. It may not be secure to reveal plaintexts to brokers.

The objective of this paper is to propose an efficient security scheme for content-based pub-sub systems such that the infrastructure can perform content-

based routing without knowing the plaintexts of publications/subscriptions. By transforming interval-matching into prefix-matching and applying a prefix-preserving encryption algorithm, we propose a novel scheme to achieve the objective.

The paper is organized as the follows, in Section 2, we define the problem formally and describe the setting for which our scheme is proposed. Section 3 surveys the related work and discusses possible solutions based on well-known mechanisms. In Section 4, we present a simple scheme that can only support exact-matching as the predicate function straightforwardly, and then in Section 5, we present a scheme that efficiently supports interval-matching as a predicate function. First we show how an interval-matching problem can be transformed into a set of prefix-matching problems. Then the prefix-preserving encryption algorithm is presented and the corresponding decryption algorithm is provided. At the end of the section, we describe a protocol that enables content-based routing without revealing the plaintexts to the infrastructure. Section 6 analyzes the security of the proposed scheme, followed by Section 7 where we discuss how to improve the security of our scheme. We then conclude the paper in Section 8.

2 Problem description

A content-based pub-sub system consists of subscribers, publishers, brokers, an event schema and predicate functions expressing subscribers' interest. While the concepts like subscribers, publishers and brokers are quite clear, we give a more formal description of the event schema and predicate functions.

Throughout this paper, attributes are denoted by capital letters, such as I, J, K, and attribute values are denoted by lowercase letters such as i, j, k. An event schema is a tuple of attributes $\{I_1, I_2, ..., I_n\}$, and an event is an instance of the event schema. There can be additional content associated with the event that is not included in the event schema. We call this information the *payload* of the event. Because the payload is not needed for the content-based routing, in this paper we will only be concerned with the attributes of the event that are included in the event schema. In the stock quote example, the attributes are I_1 = symbol (of string), I_2 = price (of dollar) and I_3 = volume (of integer). The event schema is defined as the tuple (symbol, price, volume). An event is an instance of the event schema, e.g., ("ATT", \$50, 3000). A predicate function is a boolean function that returns true when the input (an event) is in a subset of the event schema such as (i_1="ATT", $\$20 \leq i_2 \leq \$80$, $2000 \leq i_3 \leq 4000$ or $20000 \leq i_3 \leq 40000$).

In general, we can assume that content-based pub-sub systems allow each predicate to be *range-based*, composed of *intervals* in the underlying domain of the predicate [21, 22]. By decomposing a subscription with multiple such ranges into multiple subscriptions consisting of single ranges we can see that it is sufficient only to consider intervals, albeit at a cost of more subscriptions. And even attributes such as name, not typically thought of as numerical, can be indexed and therefore linearized in some fashion. Thus we restrict our pub-sub model in such a way that the predicate function can only be *exact-matching* or *interval-matching* [1]. Interval-matching is defined as a boolean function $f_{[a,b]}(x)$, which returns true

[1] Those more complex predicate functions usually can be converted into exact-matching, though it might not be efficient to do so.

if and only if $x \in [a, b]$. Because computers can handle only inherently finite and discrete attribute values, one can assume without loss of generality x, a and b are all nonnegative integers. Exact-matching is a special case of interval-matching in which a is equal to b. We will only be concerned with interval-matching or exact-matching as a predicate function.

In this paper we are not attempting to deal with a pub-sub system that has a universe of subscribers/publishers. Instead, we are targeting private pub-sub systems over public networks. Our scheme is proposed for those pub-sub systems in which publishers and subscribers are able to share some secret. This is realistic when publishers and subscribers are all from a same organization, e.g., a corporation.

The issue of confidentiality protection is that the plaintexts of publications and subscriptions need to be kept confidential from the infrastructure to counter against eavesdropping, and the infrastructure must still be able to efficiently make correct routing decisions.

3 Related work

Several approaches based on well-known mechanisms have been discussed in [26]. A potential technique that can provide information confidentiality is computing with encrypted data [14, 2]. However, an expensive protocol between the publishers and subscribers is needed. A closely related topic to subscription confidentiality is Private Information Retrieval (PIR) [11, 12]. PIR mechanisms allow clients to access entries in a database without revealing which entries they are interested in. PIR and subscription confidentiality are very closely related [26]. All PIR-based schemes move some filtering operations from the database to the user, which implies more communication overhead as well as subscriber-side computation load. Therefore, there will be challenges regarding performance and efficiency.

Many researchers have investigated the problem of zero-knowledge proof [15, 17]. In particular, several protocols have been proposed [18, 6, 5] to prove that a committed number lies in an interval without actually revealing the number. But the performance in terms of bandwidth and CPU power makes these protocols unattractive to the pub-sub systems. Moreover, it can only be used to preserve information confidentiality. Secure multi-party computation has also been intensely studied [29, 16, 8]. It would require a high computation and communication overhead if secure multi-party computation is used to solve the problem of preserving confidentiality in content-based pub-sub systems.

Aforementioned techniques are often computationally intensive and require a large amount of communication overhead, which could be prohibitively expensive to carry out in content-based pub-sub systems. Also, most of these techniques can either only preserve information confidentiality or only preserve subscription confidentiality. In contrast, the efficient scheme presented in this paper can preserve both information confidentiality and subscription confidentiality.

Opyrchal *et al.* have studied the "secure end-point delivery" problem in content-based pub-sub system, i.e., how to make sure only authorized subscribers can access the data [19]. In a content-based system, every event can potentially have a different set of interested subscribers. When the number of subscribers

is large, it is infeasible to setup static security groups for every possible subset. Their approach is to use a dynamic caching scheme, where the last-hop broker and subscribers cache subgroup keys. The "secure end-point delivery" problem and the problem addressed by this paper are complementary to each other.

4 A simple scheme

Since we are targeting private pub-sub systems over public networks, it is possible for publishers and subscribers to share some secret keys. Many group key distribution protocols have been proposed in the literature [7, 27, 20]. For example, ELK [20] can be used for publishers and subscribers to share secret keys. Since the key distribution problem is orthogonal to the problem addressed by this paper, we do not elaborate it. In a private pub-sub system, the total number of publishers/subscribers will not be too large. Therefore, key distribution is feasible.

A simple cryptographic scheme can be as follows. With a secret key, denoted by κ, publishers and subscribers construct a one-to-one mapping F_κ : $(i_1, i_2, ..., i_n) \rightarrow (j_1, j_2, ..., j_n)$. We call $(j_1, j_2, ..., j_n)$ the *encrypted event* of event $(i_1, i_2, ..., i_n)$ in the remainder of the paper, and the mapping F_κ is termed as the encrypt function. Before sending out an event, a publisher encrypts it by applying F_κ. Therefore, a published event will be F_κ(event). A subscriber generates a set of encrypted events according to his/her interest and subscribes to the network. As publishers and subscribers share the same secret key, the encrypted events they generate will match as long as the original events match. Therefore, encrypted events can be routed correctly to subscribers by brokers while the plaintexts of publications/subscriptions are protected. After receiving an encrypted event, subscribers can decrypt the event, because F_κ is a one-to-one mapping.

One straightforward way to build the mapping F_κ would be to pad attributes into a multiple of 64-bit or 128-bit fields that is suitable for applying standard encryption algorithms like DES or AES [13]. Then, by applying the encryption algorithm, we can obtain encrypted attributes. Thus a mapping F_κ is established.

The drawback of this simple scheme is that it can only support exact-matching as a predicate function. Interval-matching must be transformed into a set of exact-matchings, potentially generating an intractable number of comparisons per event. Instead, what is needed is a scheme that efficiently supports interval-matching as a predicate function, yielding reduced computation and communication burdens. In the remainder of this paper, such a scheme will be proposed and discussed.

5 An efficient scheme

5.1 Transforming interval-matching into prefix-matching

We have seen that it is very important to efficiently support interval-matching as a predicate function. In this section, we will transform interval-matching into prefix-matching. Prefix-matching has been used widely in networks and databases. The transformation is based on the fact that an arbitrary interval can be converted into a union of *prefix ranges*, where a prefix range is one that can be expressed by a prefix [25]. For example, the interval [32, 111], the 8-bit binary representation

of which is [00100000, 01101111], can be represented by a set of prefixes {001∗, 010∗, 0110∗}. Throughout this paper, the notation ∗ is used to denote an arbitrary suffix. To verify that a number is in the interval is equivalent to check that the number matches any of those prefixes in the set. For example, 37 (00100101 in binary) is in the interval as it matches prefix 001∗, while 128 (10000000 in binary) is not in the interval since it matches none of those three prefixes.

Let n denote the length of the binary representation of the data, and let p_n denote the number of prefixes needed to represent an interval. We have the following theorem on the upper bound of p_n.

Theorem 1 *For any interval* $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ $(n \geq 2)$, $p_n \leq 2(n-1)$.

The proof of this theorem is presented in Appendix A. Note that for interval $[1, 2^n-2]$, it can be easily verified that p_n is equal to $2(n-1)$. Therefore, the upper bound is tight. In the simple scheme mentioned in the previous section, interval-matching must be transformed into a set of exact-matchings. The upper bound of the number of exact-matchings needed for an interval $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ is 2^n. Thus the approach we present in this section is much more efficient than the simple scheme mentioned in previous section.

In Figure 1 we present a recursive algorithm to generate the set of prefixes for a given interval $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$.

1. Starting from $k = 1$, find the most significant bit, numbered k, for which $a_k < b_k$.
2. If k is not found, i.e., for all $1 \leq i \leq n$, $a_i = b_i$, then the interval can be denoted by prefix $a_1a_2\cdots a_n$. Return $a_1a_2\cdots a_n$.
3. If for all $k \leq i \leq n$, $a_i = 0$ and $b_i = 1$, then return $a_1a_2a_{k-1}*$ (return $*$ if $k = 1$).
4. Transform interval $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ into $[a_1\cdots a_{k-1}0a_{k+1}\cdots a_n, a_1\cdots a_{k-1}011\cdots 1] \cup [a_1\cdots a_{k-1}100\cdots 0, a_1\cdots a_{k-1}1b_{k+1}\cdots b_n]$.
5. Run this algorithm with interval $[a_{k+1}\cdots a_n, 11\cdots 1]$ as input, concatenate $a_1\cdots a_{k-1}0$ before all the returned prefixes. Then run this algorithm with interval $[00\cdots 0, b_{k+1}\cdots b_n]$ as input, concatenate $a_1\cdots a_{k-1}1$ before all the returned prefixes. Return all the prefixes.

Figure 1: The Algorithm for transforming interval $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ into prefixes

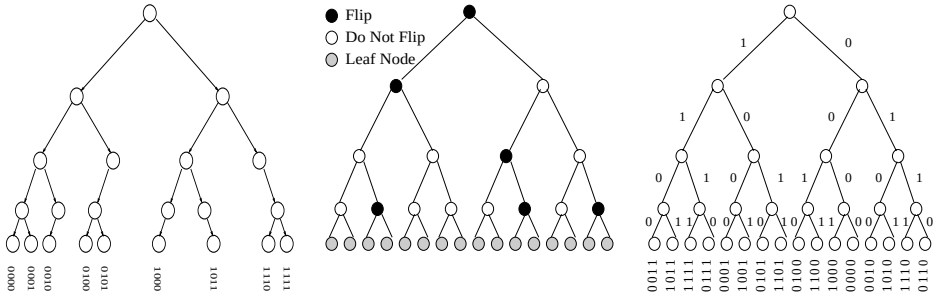
We have seen that matching an interval based on a set of prefix-matchings is both simple and efficient. Therefore prefix-preserving encryption/decryption algorithms can be used to efficiently support interval-matching as a predicate function while preserving the confidentiality of the pub-sub system.

5.2 Prefix-preserving encryption/decryption

After transforming interval-matching into prefix-matching, we need a prefix-preserving encryption/decryption scheme, so that brokers will be able to route encrypted publications based on encrypted subscriptions. We apply an encryption scheme

proposed by Xu *et al.* [28] for prefix-preserving IP address anonymization. First we introduce a formal definition of prefix-preserving encryption.

Definition 1 (Prefix-preserving encryption) (adapted from [28]) *We say that two n-bit numbers $a = a_1a_2\cdots a_n$ and $b = b_1b_2\cdots b_n$ share a k-bit prefix $(0 \leq k \leq n)$, if $a_1a_2\cdots a_k = b_1b_2\cdots b_k$, and $a_{k+1} \neq b_{k+1}$ when $k < n$. An encryption function E is defined as a one-to-one function from $\{0,1\}^n$ to $\{0,1\}^n$. An encryption function E is said to be prefix-preserving, if, given two numbers a and b that share a k-bit prefix, $E(a)$ and $E(b)$ also share a k-bit prefix.*



(a) plaintext tree (b) encryption function (c) ciphertext tree

Figure 2: An example of prefix-preserving encryption

It is helpful to consider a geometric interpretation of prefix-preserving encryption. If a plaintext can take any value of a n-bit number, the entire set of plaintexts can be represented by a complete binary tree of height n. This is called the *plaintext tree*. Each node in the plaintext tree (excluding the root node) corresponds to a bit position, indicated by the height of the node, and a bit value, indicated by the direction of the branch from its parent node. Figure 2(a) shows a plaintext tree (using 4-bit plaintexts for simplicity).

A prefix-preserving encryption function can be viewed as specifying a binary variable for each non-leaf node (including the root node) of the plaintext tree. This variable specifies whether the encryption function "flips" this bit or not. Applying the encryption function results in the rearrangement of the plaintext tree into a *ciphertext tree*. Figure 2(c) shows the ciphertext tree resulting from the encryption function shown in Figure 2(b). Note that an encryption function will, therefore, consist of $2^n - 1$ binary variables.

A general form of prefix-preserving encryption function is presented in [28]. Let f_i be a function $\{0,1\}^i$ to $\{0,1\}$, for $i = 1, 2, \cdots, n-1$ and f_0 is a constant function. Given a plaintext $a = a_1a_2\cdots a_n$, the ciphertext $a'_1a'_2\cdots a'_n$ will be computed by the algorithm given in Figure 3. According to Theorem 1 (canonical form theorem) in [28], the algorithm given in Figure 3 is a prefix-preserving encryption algorithm.

Since originally the prefix-preserving encryption algorithm was proposed for IP address anonymization, the corresponding decryption algorithm was not presented. Here we present a decryption algorithm for the prefix-preserving encryption algorithm. Given a ciphertext $a'_1a'_2\cdots a'_n$, the plaintext $a_1a_2\cdots a_n$ can be

1. Compute a'_i as $a_i \oplus f_{i-1}(a_1 a_2 \cdots a_{i-1})$, where $\oplus$ stands for the exclusive-or operation, for $i = 1, 2, \cdots, n$.
2. Return $a'_1 a'_2 \cdots a'_n$.

Figure 3: Prefix-preserving encryption algorithm

computed by the algorithm given in Figure 4.

1. Initially we have f_0. Let $i = 1$.
2. Compute a_i as $a'_i \oplus f_{i-1}(a_1 a_2 \cdots a_{i-1})$.
3. Let $i = i + 1$. If $i \leq n$, then go to step 2. Otherwise, return $a_1 a_2 \cdots a_n$.

Figure 4: Decryption algorithm

In [28], the prefix-preserving encryption scheme is defined as instantiating functions f_i with cryptographically strong stream ciphers or block ciphers as follows:

$$f_i(a_1 a_2 \cdots a_i) := \mathcal{L}(\mathcal{R}(\mathcal{P}(a_1 a_2 \cdots a_i), \kappa)) \quad (1)$$

where $i = 0, 1, \cdots, n-1$ and $\mathcal{L}$ returns the "least significant bit". Here $\mathcal{R}$ is a pseudorandom function or a pseudorandom permutation (i.e., a block cipher), and $\mathcal{P}$ is a padding function that expands $a_1 a_2 \cdots a_i$ into a longer string that matches the block size of $\mathcal{R}$. κ is the cryptographic key used in the pseudorandom function $\mathcal{R}$. Its length should follow the guideline specified for the pseudorandom function that is actually adopted.

5.3 The protocol

With the prefix-preserving encryption/decryption algorithms, denoted by E and D respectively, the protocol runs as follows. When a publisher wants to send out an event $(i_1, i_2, ..., i_n)$, he/she encrypts each attribute with the prefix-preserving encryption algorithm. Then the encrypted event $(E(i_1), E(i_2), ..., E(i_n))$ is sent out. When a subscriber wants to subscribe, he/she computes $(P_1, P_2, ..., P_n)$ with the algorithm given in Figure 1, where P_i $(i = 1, 2, ..., n)$ is a set of prefixes representing intervals. Before subscribing, the subscriber also applies the prefix-preserving encryption to every prefix in P_i. The infrastructure can then route encrypted events correctly based on encrypted predicate functions [2]. We present our protocol formally as follows.

When a publisher wants to send out an event $(i_1, i_2, ..., i_n)$,

1. Each attribute is encrypted with the prefix-preserving encryption algorithm.
2. The encrypted event $(E(i_1), E(i_2), ..., E(i_n))$ is published.

[2]Several researchers have studied the problem of evaluating a possibly large number of predicates against message like data (events) in the domain of content-based forwarding for publish-subscribe systems. For this problem, various forms of decision trees and indexing structures for subscriptions have been proposed (see, e.g., [3, 10]). Content-based routing protocols have also been proposed (see, e.g., [4, 9]). The dispatching of encrypted events in our proposed scheme can be achieved by naturally applying these techniques.

When a subscriber wants to subscribe his/her predicate, which is a set of interval-matching functions in the attribute sets,

1. The intervals that are used for the predicate against each attribute are translated into a set of prefixes, P_i, $i = 1, 2, ..., n$, using the algorithm given in Figure 1.

2. The prefix-preserving encryption algorithm is applied to each prefix in the tuple $(P_1, P_2, ..., P_n)$, resulting in a tuple of encrypted prefix sets $(P'_1, P'_2, ..., P'_n)$.

3. The encrypted prefix sets are subscribed to the broker network.

Upon receipt of an encrypted event, a broker checks if the encrypted event satisfies a certain encrypted subscription by checking if attributes in the event match any of the prefixes in the corresponding prefix set. After receiving an event, subscribers apply the decryption algorithm to recover the original value of the attributes.

The encryption function can be performed quickly as it only involves n symmetric key cryptographic operations, and these n operations can be done in parallel. The decryption also only involves n symmetric key cryptographic operations, though these n operations have to be done serially. The prefix-matching can be done very efficiently, thus the routing decision by brokers can be made quickly. Therefore, high performance in the content-based pub-sub can be achieved. Also, the prefix-preserving encryption scheme preserves the bit length of the plaintext as well, hence the ciphertext requires no more space than the plaintext.

The prefix-preserving technique leaks information, i.e., the security of the proposed scheme is limited for the purpose of efficiency. In Section 6 we analyze the security of prefix-preserving encryption, and in Section 7 several techniques are proposed to further improve the security of the system.

6 Security analysis

Since we assume that the connections among publishers and subscribers can be over public networks, an attacker can eavesdrop the connections and download the encrypted messages (publications or subscriptions). In this section we analyze the security of the proposed encryption/decryption scheme. It has been proved that with the instantiating functions as (1) our prefix-preserving encryption scheme is indistinguishable from a *random prefix-preserving function*, a function uniformly chosen from the set of all prefix-preserving functions when the adversaries are assumed to be computationally bounded. This is elaborated in [28]. Moreover, as mentioned in Section 5.2, when plaintexts can take any value of a n-bit number, the prefix-preserving encryption function consists of $2^n - 1$ binary variables. Therefore, we have a key of 2^{2^n-1} possibilities. For example, when n is only 16, the number of possible keys is 2^{65535}. Therefore, the key κ in (1) can be chosen to be sufficiently long such that it is impractical for eavesdroppers to try each possible key to compromise our scheme.

In the remainder of this section, we discuss another possible way in which our scheme may be attacked. An eavesdropper is assumed to have compromised (gain full knowledge to) certain number of ⟨plaintext, ciphertext⟩ pairs

through means other than compromising the key (i.e., the known plaintext attack model). Then he/she will be able to infer information from other ciphertexts by prefix-matching, because the encryption is prefix-preserving. For example, if an eavesdropper knows ⟨plaintext, ciphertext⟩ pair $\langle a_1 a_2 \cdots a_n, a'_1 a'_2 \cdots a'_n \rangle$, then given another ciphertext $a'_1 a'_2 \cdots a'_{k-1} \overline{a'_k} b_{k+1} \cdots b'_n$, he/she knows the k-bit prefix of the plaintext should be $a_1 a_2 \cdots a_{k-1} \overline{a_k}$. Note that if an eavesdropper knows one ⟨plaintext, ciphertext⟩ pair $\langle a_1 a_2 \cdots a_n, a'_1 a'_2 \cdots a'_n \rangle$, then he/she should also know the ⟨plaintext, ciphtertext⟩ pair $\langle a_1 a_2 \cdots \overline{a_n}, a'_1 a'_2 \cdots \overline{a'_n} \rangle$. Therefore, an eavesdropper always knows an even number of ⟨plaintext, ciphertext⟩ pairs.

Suppose an eavesdropper knows 2 pairs of ⟨plaintext, ciphertext⟩. Given a random ciphertext, let $A(n)$ denote the average length of the prefix that can be inferred by prefix-matching, where n is the length of the binary representation of the data. The probability that the k-bit prefix of the plaintext can be inferred is $\frac{1}{2^k}$, for $1 \leq k \leq n-1$, while for $k = n$, the probability is $\frac{2}{2^n}$. Therefore, $A(n) = \sum_{i=1}^{n-1} \frac{i}{2^i} + \frac{2n}{2^n} = \sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}} < 2$. In other words, on the average an eavesdropper can infer no more than 2 bits from a random ciphertext, if he/she knowns 2 pairs of ⟨plaintext, ciphertext⟩.

We also analyze the situation that an eavesdropper knows $2k$ ($k > 1$) pairs of ⟨plaintext, ciphertext⟩. This is presented in Appendix B. In summary, when $n \to \infty$, given a ciphertext, the average length of the prefix that can be inferred is bounded by $\log_2 k + 2$ based on numerical results. So the prefix information an eavesdropper can obtain by comparing a ciphtertext against a few pairs of ⟨plaintext, ciphertext⟩ is limited. Therefore, we claim that our scheme is secure even if a few pairs of ⟨plaintext, ciphertext⟩ are known by an eavesdropper. It is potentially dangerous when the number of ⟨plaintext, ciphertext⟩ pairs accumulates at the eavesdropper side, since it will lead to more prefix information of encrypted messages to be revealed. The improved system discussed in the next section will update the shared key among publishers and subscribers periodically. Therefore, ⟨plaintext, ciphertext⟩ pairs known by an eavesdropper will become obsolete when the shared key for the encryption/decryption scheme is changed.

7 Improving the security

From the previous section we can see that it is possible for eavesdroppers to partially reveal the content of publications/subscriptions. Some techniques are discussed in this section to make the system more secure. The weakness of the system depends on the amount of information the eavesdropper can collect from publications and subscriptions. Hence, we can reinforce the confidentiality by limiting the amount of information an eavesdropper can get against a specific encryption function. This can be achieved in two ways. First, we can use different keys to encrypt different attributes, thus prevent an eavesdropper from aggregating the information from different attributes. Second, the encryption/decryption functions can be invalidated and renewed at constant time intervals. Therefore, the amount of information disseminated by each encryption function is reduced. Moreover, the knowledge on a particular encryption function will be obsolete after the function becomes invalid. Thus, the lifetime of such knowledge is bounded by the time intervals.

In the following we discuss the aforementioned two techniques together in de-

tail, though they can work separately. The construction of encryption/decryption functions is equivalent to the generation of secret keys. Therefore, it is sufficient for us to discuss how to generate secret keys for the encryption/decryption functions at time intervals. Suppose there are n attributes in an event schema. To achieve the maximum security, a different key is needed to construct encryption/decryption functions for each attribute. Publishers and subscribers share n secret keys $\kappa_1, \kappa_2, ..., \kappa_n$ and an initial vector IV by using a group key distribution protocol [7, 27, 20]. Session keys $E^{(i)}_{\kappa_1}(IV), E^{(i)}_{\kappa_2}(IV), ..., E^{(i)}_{\kappa_n}(IV)$ are generated for the attributes in the ith time interval, where the notation $E^{(i)}_{\kappa}(IV)$ means applying some sort of encryption E (e.g. DES or AES [13]) for i times. Therefore, the secret keys for each time interval are represented below by each row.

$$
\begin{gathered}
E^{(1)}_{k_1}(IV), E^{(1)}_{k_2}(IV), ..., E^{(1)}_{k_n}(IV) \\
E^{(2)}_{k_1}(IV), E^{(2)}_{k_2}(IV), ..., E^{(2)}_{k_n}(IV) \\
\vdots \\
E^{(i)}_{k_1}(IV), E^{(i)}_{k_2}(IV), ..., E^{(i)}_{k_n}(IV) \\
\vdots
\end{gathered}
$$

The proposed method is similar to the technique used in generating key stream in the stream ciphers. However, it does not combine the encrypted blocks into a key stream. Instead, the blocks are used as session keys for different time intervals in the pub-sub system. The use of the session keys will also prevent an eavesdropper from breaking the master keys even with the knowledge of some session keys.

Because we are changing keys at constant time intervals, we must handle the synchronization among subscribers and publishers. One possible solution is to let publishers/subscribers independently update the keys based on time of day. This requires synchronizing the clocks of all publishers/subscribers. Sometimes this is difficult to achieve, especially in a wide-area network. In order to solve this problem we introduce another entity called the *synchronization server*, which will issue a key changing signal at constant time intervals. The synchronization server can be viewed as a publisher in the network. All the other entities subscribe to it for the key changing signal. Upon receipt of the signal, publishers and subscribers update the session keys, and subscribers regenerate their subscriptions.

The problem is that the key changing signal will not arrive at the publishers/subscribers at the same time. To ensure that events will be correctly routed to subscribers, a publisher will encrypt the event with both the new and the old keys for a certain time period Δt_1. Then the publisher discards the old keys and starts using solely the new keys. Let the maximum difference among the propagation delays from synchronization server to other entities be Δt_{max}, and let the maximum time needed for a new subscription to reach every broker be Δt_{sub}. Then Δt_1 should be no less than Δt_{max} plus Δt_{sub}. We assume both Δt_{max} and Δt_{sub} are measurable or at least can be estimated. Upon receipt of a key changing signal, subscribers should subscribe the new encrypted subscriptions immediately and unsubscribe the old subscriptions after a certain period of time Δt_2, which

should be no less than Δt_{max}. During Δt_2, subscribers should try both the old and the new decryption functions (We are assuming there is some signature in the encrypted event, so subscribers will know which key is valid for the encrypted event received.).

Changing keys at time intervals will secure the system in a very strong sense, but it introduces additional traffic as publishers need to publish duplicate events in a short time period and subscribers need to re-subscribe requests. It also introduces more computation, because in a short period of time publishers need to encrypt events twice, subscribers need to try two different sets of keys for every event and brokers need to do more prefix-matchings. Therefore the change must not be performed too frequently. Also using a synchronization server may create a vulnerable point. Therefore, there is a need to build the synchronization server with high level of security, or to deploy multiple synchronization servers to provide backup for the signaling service.

8 Conclusions

This paper discusses concerns about protecting sensitive information of events (publications) and predicate functions (subscriptions) from eavesdroppers in content-based sub-pub systems. Events and predicate functions need to be encrypted, while brokers should be able to efficiently make correct routing decisions based on encrypted events and predicate functions. This issue is addressed by a novel publication-subscription encryption/decryption scheme. This scheme efficiently supports interval-matching as predicate functions by transforming interval-matching into prefix-matching and using a prefix-preserving encryption algorithm. The security of prefix-preserving encryption is analyzed, and several techniques are proposed to further improve the security of the system. The paper shows that this scheme preserves confidentiality while maintaining efficiency.

A major contribution of this paper is the efficient scheme supporting interval-matching on encrypted data. This scheme can also be used in some other applications. For instance, it is desirable to store data on storage servers in encrypted form to reduce security and privacy risks [24]. Our scheme can be used to efficiently support range-based searching on encrypted data.

References

[1] Gryphon: Publish/subscribe over public networks. http://www.research.ibm.com/gryphon/ Gryphon/gryphon.html. IBM T.J. Watson Research Center.

[2] Martin Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. In *Proceedings of the 19th ACM Annual Symposium on Theory of Computing*, May 1987.

[3] Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley, and Tushar Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, May 1999.

[4] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999.

[5] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Proceedings of EUROCRYPT'00*, pages 431–444, 2000.

[6] E. Brickell, D. Chaum, I. Damgard, and Van de Graaf. Gradual and verifiable release of a secret. In *Proceedings of CRYPTO'87*, pages 156–166, 1987.

[7] R. Canetti, J. Garay, G. Itkis, D. Miccianancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *Proceedings of INFOCOM'99*, March 1999.

[8] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Israel, 1995.

[9] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[10] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM'03*, August 2003.

[11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 41–35, October 1995.

[12] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Proceedings of Advances in Cryptology – EUROCRYPT'00*, 2000.

[13] J. Daemen and V. Rijmen. AES proposal: Rijndael. Technical report, Computer Security Resource Center, National Institute of Standards and Technology, February 2001.

[14] Joan Feigenbaum. Encrypting problem instances, or ..., can you take advantage of someone without having to trust him? In *Proceedings of CRYPTO'85*, 1986.

[15] Zvi Galil, Stuart Haber, and Mordechai Yung. A private interactive test of a boolean predicate and minimum-knowledge of public-key cryptosystems. In *Proceedings of the 26th IEEE Annual Symposium on Foundations of Computer Science*, pages 360–371, 1985.

[16] Oded Goldreich. Secure multi-party computation. Working Draft, 2000.

[17] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th ACM Annual Symposium on Theory of Computing*, pages 291–304, 1985.

[18] W. Mao. Guaranteed correct sharing of integer factorization with off-line share-holders. In *Proceedings of Public Key Cryptography'98*, pages 27–42, 1998.

[19] Lukasz Opyrchal and Atul Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of the 10th USENIX Security Symposium*, August 2001.

[20] A. Perrig, D. Song, and D. Tygar. ELK, a new protocol for efficient large-group key distribution. In *Proceedings of the 22st IEEE Symposium on Research in Security and Privacy*, May 2001.

[21] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems*, July 2002.

[22] Anton Riabov, Zhen Liu, Joel L. Wolf, Philip S. Yu, and Li Zhang. New algorithms for content-based publication-subscription systems. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems*, May 2003.

[23] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, September 1997.

[24] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 21st IEEE Symposium on Research in Security and Privacy*, May 2000.

[25] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proceedings of ACM SIGCOMM'98*, September 1998.

[26] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35)*, January 2002.

[27] C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(6):16–30, February 2000.

[28] Jun Xu, Jinliang Fan, Mostafa H. Ammar, and Sue B. Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, November 2002.

[29] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science*, November 1982.

Appendix

A Proof of Theorem 1

In this appendix, we offer a proof of Theorem 1 (introduced in Section 5.1), which gives the upper bound of the number of prefixes needed to represent an interval. We will use the same notations as in Section 5.1.

Lemma 1 *For any interval* $[0, a_1a_2\cdots a_n]$, $p_n \leq n$.
Proof: We prove it by induction on n. The conclusion trivially holds for $n = 1$. Suppose the conclusion also holds for $n = k$.
We now prove the lemma for $n = k + 1$. If a_1 is equal to 0, then according to induction hypothesis, we have $p_n \leq k$. If for all $1 \leq i \leq k + 1$, $a_i = 1$, then the interval can be represented by prefix $*$, i.e., $p_n = 1$. Otherwise, the interval $[0, a_1a_2\cdots a_{k+1}]$ can be represented by $[0, 011\cdots1] \cup [100\cdots0, 1a_2\cdots a_{k+1}]$. $[0, 011\cdots1]$ can be represented by prefix $0*$. According to induction hypothesis, we need at most k prefixes to represent interval $[100\cdots0, 1a_2\cdots a_{k+1}]$. Hence for $n = k + 1$, we have $p_n \leq k + 1$. □

Lemma 2 *For any interval* $[a_1a_2\cdots a_n, 11\cdots1]$, $p_n \leq n$.
The proof is omitted here, because it is similar to the proof of Lemma 1.

Theorem 1 *For any interval* $[a_1a_2\cdots a_n, b_1b_2\cdots b_n]$ $(n \geq 2)$, $p_n \leq 2(n-1)$.
Proof: We prove it by induction on n.
For $n = 2$, if $a_1a_2 = 00$ or $b_1b_2 = 11$, then according to Lemma 1 and Lemma 2, we have $p_n \leq 2$. If $a_1a_2 = b_1b_2 = 01$ or 10, then $p_n = 1$. If $a_1a_2 = 01$ and $b_1b_2 = 10$, then $p_n = 2$. So the conclusion holds for $n = 2$.
Suppose the conclusion also holds for $n = k$.
We now prove the theorem for $n = k + 1$. If $a_1 = b_1$, then according to induction hypothesis, we have $p_n \leq 2(k - 1)$. Otherwise, we have $a_1 = 0$ and $b_1 = 1$. If for all $1 \leq i \leq k+1$, $a_i = 0$ and $b_i = 1$, then the interval can be represented by prefix $*$, i.e., $p_n = 1$. The interval $[a_1a_2\cdots a_{k+1}, b_1b_2\cdots b_{k+1}]$ can be represented by $[0a_2\cdots a_{k+1}, 011\cdots1] \cup [100\cdots0, 1b_2\cdots b_{k+1}]$. According to Lemma 2, we need at most k prefixes to represent interval $[0a_2\cdots a_{k+1}, 011\cdots1]$, and according to Lemma 1, we need at most k prefixes to represent interval $[100\cdots0, 1b_2\cdots b_{k+1}]$. Hence for $n = k + 1$, we have $p_n \leq 2k$. □

B Known plaintext attack

In this appendix, we analyze the information an eavesdropper can obtain by comparing the ⟨plaintext, ciphertext⟩ pairs known by him/her to a ciphertext. Hereafter we will assume the length of the binary representation of the data, denoted by n, is very long, i.e., $n \to \infty$. Suppose that an eavesdropper obtains $2k$ pairs of ⟨plaintext, ciphertext⟩ randomly, and the average length of the prefix he/she can obtain from a random ciphertext is A_k. Then A_k can be computed as follows.

In the first step, the eavesdropper will compare the first bit of the ciphertext with all the $2k$ pairs of ⟨plaintext, ciphertext⟩. If, among them, $2l$ pairs of

k	1	2	4	8	16
A_k	2	2.666667	3.504762	4.421077	5.377378
k	32	64	128	256	512
A_k	6.355176	7.343990	8.338377	9.335558	10.334156

Table 1: A_k by varying k

$\langle$plaintext, ciphertext$\rangle$ match the first bit of the ciphertext, then the other $2(k-l)$ pairs are not useful for further deriving prefix information. Therefore, the total prefix information the eavesdropper can obtain is the first bit plus the prefix information he/she may obtain with $2l$ pairs of $\langle$plaintext, ciphertext$\rangle$, which is equal to $1 + A_l$. Since the possibility for $2l$ pairs out of $2k$ pairs of $\langle$plaintext, ciphertext$\rangle$ to match the first bit of a ciphertext is $(1/2)^k C_k^l$, the average length of the prefix an eavesdropper can obtain is $A_k = \sum_{l=0}^{k} (1/2)^k C_k^l (1 + A_l)$.

It is obvious that A_0 is equal to 0. The remaining values of A_k can be computed inductively. For example, $A_1 = 1 + \frac{1}{2}A_1$, thus $A_1 = 2$, which is consistent with the result presented in Section 6. Table 1 shows A_k for several different values of k. Figure 5 presents the curve of A_k by varying k, which is bounded by $\log_2 k + 2$. Therefore, the prefix information that can be revealed by a few pairs of $\langle$plaintext, ciphertext$\rangle$ is limited.

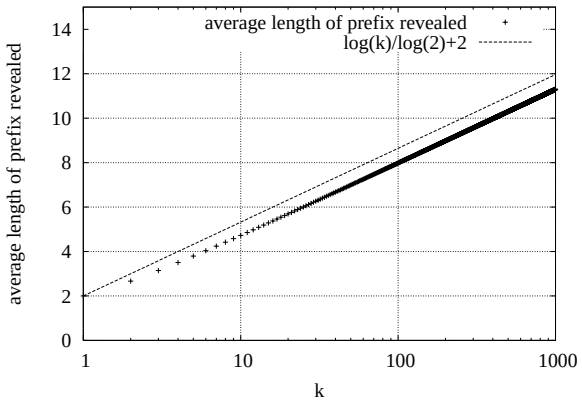


Figure 5: A_k by varying k